Requested Patent: JP10269105A

Title:

TRACE SYSTEM, RESOURCE RELEASE OMISSION DETECTION SYSTEM, AND RECORDING MEDIUM ;

Abstracted Patent: JP10269105 ;

Publication Date: 1998-10-09 ;

Inventor(s): TANIMURA MORIMASA; NISHISAKA TAKESHI ;

Applicant(s): N T T DATA TSUSHIN KK ;

Application Number: JP19980013654 19980127 ;

Priority Number(s): ;

IPC Classification: G06F11/28 ; G06F11/30 ;

Equivalents:

ABSTRACT:

PROBLEM TO BE SOLVED: To globally trace an OS(operating system) without recompiling it, and to effectively detect memory lead because of it even in a system which is made by combining components provided in binary. SOLUTION: A hooking DLL(dynamic link library) 2 hooks an API(application interface) call of an application part 1. The DLL 2 is inserted instead of an original DLL that corresponds to an API call. A processing DLL 3 executes processing that corresponds to the API call. The DLL 2 hooks a call of the part 1, sends the call information to a log filing part 4 and also transfers the call to the DLL 3. The part 4 logs the call information that is hooked by the DLL 2. A viewer 5 reads and shows log contents in the part 4.

(19)日本国特許庁（J P）　(12)公開特許公報（A）　(11)特許出願公開番号

特開平10－269105

(43)公開日　平成10年(1998)10月9日

(51)Int.Cl.[6]		識別記号	FI		
G06F	11/28	310	G06F	11/28	310A
	11/30	320		11/30	320C

審査請求　未請求　請求項の数13　OL　（全 12 頁）

(21)出願番号　特願平10－13654

(22)出願日　平成10年(1998)1月27日

(31)優先権主張番号　特願平9－13012

(32)優先日　平9(1997)1月27日

(33)優先権主張国　日本（J P）

(71)出願人　000102728
エヌ・ティ・ティ・データ通信株式会社
東京都江東区豊洲三丁目3番3号

(72)発明者　谷村　守正
東京都江東区豊洲三丁目3番3号　エヌ・ティ・ティ・データ通信株式会社内

(72)発明者　西坂　毅
東京都江東区豊洲三丁目3番3号　エヌ・ティ・ティ・データ通信株式会社内

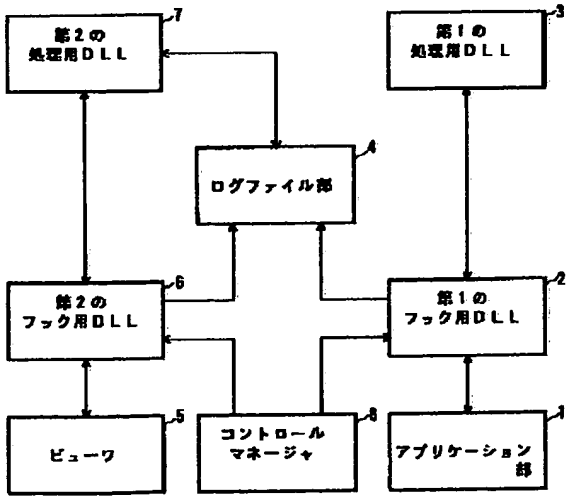
(74)代理人　弁理士　木村　満

(54)【発明の名称】　トレースシステム、リソース解放漏れ検出システム及び記録媒体

(57)【要約】

【課題】　バイナリで提供されたコンポーネントを組み合わせて作成されたシステムにおいても、リコンパイルせずにOSのグローバルなトレース及びそれによるメモリリークの効果的な検出を可能とする。

【解決手段】　フック用DLL2は、アプリケーション部1のAPI呼び出しをフックする。フック用DLL2は、API呼び出しに対応する本来のDLLの代わりに挿入される。処理用DLL3は、該当するAPI呼び出しに対応する処理を実行する。フック用DLL2は、アプリケーション部1の呼び出しをフックし、該呼び出しの情報をログファイル部4に送信するとともに、該呼び出しを処理用DLL3に転送する。ログファイル部4は、フック用DLL2でフックされた呼び出し情報をロギングする。ビューワ5は、ログファイル部4のログ内容を読み込み表示する。



【特許請求の範囲】
【請求項1】アプリケーションプログラムとのインタフェースを取るためのアプリケーションインタフェースを備えるオペレーティングシステムと、
前記オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段と、
前記アプリケーション手段による前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しを中継するとともに、該呼び出しの情報を取り込むフック手段と、
前記フック手段で取り込まれた前記呼び出しの情報を蓄積記録するログ記録手段と、を具備することを特徴とするトレースシステム。
【請求項2】前記フック手段は、前記アプリケーションインタフェースに対するアプリケーションプログラムの呼び出しを前記アプリケーションインタフェースの代わりに受ける手段と、前記呼び出しの情報を前記ログ記録手段に提供する手段と、前記呼び出しを前記アプリケーションインタフェースに転送する手段とを含むことを特徴とする請求項1に記載のトレースシステム。
【請求項3】前記ログ記録手段で記録された前記呼び出しの情報を出力する手段をさらに含むことを特徴とする請求項1又は2に記載のトレースシステム。
【請求項4】前記フック手段から前記ログ記録手段へ転送された情報の記録のオンとオフを制御するロギング制御手段をさらに含むことを特徴とする請求項1乃至3のうちのいずれか1項に記載のトレースシステム。
【請求項5】前記ロギング制御手段は、前記フック手段から前記アプリケーションインタフェースへの前記情報の転送の停止と実行を、外部操作に応答して、制御する手段をさらに含むことを特徴とする請求項4に記載のトレースシステム。
【請求項6】前記ロギング制御手段は、前記ログ記録手段に記録される前記情報の内容を制御する手段をさらに含むことを特徴とする請求項4又は5に記載のトレースシステム。
【請求項7】前記フック手段は、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しのうち、前記オペレーティングシステムのリソース割り当てに関するリソース割り当て情報を取り込む手段を備えることを特徴とする請求項1乃至6のいずれか1項に記載のトレースシステム。
【請求項8】前記ログ記録手段により、記録されたリソースの割り当てと解放の対を検出し、割り当てと解放の一方のみを検出し、他方を検出できない時に、その旨を報知する対検出手段をさらに含むことを特徴とする請求項7に記載のトレースシステム。
【請求項9】オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段と、
前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムから前記オペレーティングシステムのアプリケーションインタフェースに送られるリソースの割り当てと解放の要求に関するリソース割当情報を取り込むフック手段と、
前記フック手段で取り込まれた前記リソース割当情報を蓄積記録するログ記録手段と、を具備することを特徴とするリソース解放漏れ検出システム。
【請求項10】前記ログ記録手段により記録されたリソースの割り当てと解放の対を検出し、割り当てと解放の一方のみを検出し、他方を検出できない時に、その旨を報知する対検出手段をさらに含むことを特徴とする請求項9に記載のリソース解放漏れ検出システム。
【請求項11】前記リソースはメモリである、ことを特徴とする請求項9又は10に記載のリソース解放漏れ検出システム。
【請求項12】コンピュータを、
アプリケーションプログラムとのインタフェースを取るためのアプリケーションインタフェースを備えるオペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段、
前記アプリケーション手段による前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しを中継するとともに、該呼び出しの情報を取り込むフック手段と、
前記フック手段で取り込まれた前記呼び出しの情報を蓄積記録するログ記録手段、として機能させるためのプログラムを記録したコンピュータで読み取り可能な記録媒体。
【請求項13】コンピュータを、
オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段、
前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムから前記オペレーティングシステムのアプリケーションインタフェースに送られるリソースの割り当てと解放の要求に関するリソース割当情報を取り込むフック手段、
前記フック手段で取り込まれた前記リソース割当情報を蓄積記録するログ記録手段、として機能させるためのプログラムを記録したコンピュータで読み取り可能な記録媒体。
【発明の詳細な説明】
【0001】
【発明の属する技術分野】この発明は、アプリケーションプログラムの開発或いはチェック等に際し、システム全体のグローバルなトレースを行うためのトレースシステム及びリソースの解放漏れを検出するためのシステムに係り、特にシステム全体のパフォーマンスチューニングやメモリリークの効果的な検出に好適なトレースシス

テム及びリソース解放漏れ検出システムに関する。

【0002】

【従来の技術】オペレーティングシステム（OS: Operating System〜以下、「ＯＳ」と称する）において動作するアプリケーションプログラム及びシステムプログラム等のプログラムにおいては、ＯＳのアプリケーションインタフェース（API: Application Program Interface：以下、「ＡＰＩ」と称する）を使用して、メモリ割り当てを行う。

【0003】プログラムに対して割り当てられたメモリは、メモリが不要となった時点で、解放される。一般にメモリ等のリソース（資源）には、限りがあるため、利用済みのメモリが解放されなければ、利用できるメモリの量が低減し、プログラムを正常に実行できなくなる。ところが、プログラムの不備により、一旦割り当てられたメモリが不要になった時点でも解放しないことがある。このように、不要となったメモリを解放せずに割り当てられたままにしておくことを、メモリリーク（メモリ解放漏れ）と称する。このメモリリークを放置すると、前述のように、利用できるメモリの量が低減し、プログラムを正常に実行できなくなる。

【0004】

【発明が解決しようとする課題】システムが、大きくなればなるほど、上述のメモリリークが発生してしまう可能性が高くなる。一方、メモリリーク自体は、小さなメモリリークほど、開発における各種チェック工程で見落とす可能性が高い。このように、開発工程でメモリリークが見落とされた場合、システムの最終確認フェーズ、すなわち連続運転試験において、やっと発見されることになる。しかも、メモリリークが、プログラムのどの部分で発生しているかを見分けるためには、ソースレビューなどの地道な方法で行うしかなかった。

【0005】特に、ウィンドウズＮＴ（Windows NT）及びウィンドウズ９５（Windows 95）（「Windows」は登録商標）等のＯＳは、複数のプロセスからなり、個々のプロセスは、オブジェクトリンク及び埋め込み（以下、「ＯＬＥ」）機能を利用するＯＬＥサーバオブジェクトであるＯＬＥコントロール（以下、「ＯＣＸ」）、並びにダイナミックリンクライブラリ（以下、「ＤＬＬ」）等を１つ以上含んでいる。なお、ＯＣＸは、最近では、Active Xコントロールと称されるコンポーネントとして位置付けられている。

【0006】このようなＯＳでは、プロセス単位でしかリソースを管理していないため、プロセスに属しているＤＬＬやＯＣＸ等に問題があったとしても、それらを切り分けることは不可能である。このため、場合によっては、システムの全域にわたってメモリリークの原因を調査する必要がある。

【0007】このように、アプリケーションプログラム及びシステムプログラムの開発に際して、メモリリークが、一旦、発生してしまうと、その解決に膨大な時間を費やすことが少なくない。

【0008】通常、メモリリークの検出等を目的として、プログラムのトレースを行う場合には、特別なオブジェクト（ここで、オブジェクトとは、ＯＢＪファイル又はＣ＋＋におけるオブジェクトを指している）をリンクするなどの手法を用いる。このため、プログラムのトレースのために、ソースコードからのコンパイル及びリンクを再度やり直す必要がある。しかしながら、この方法では、最近、広く用いられているＯＣＸ及びActiveXコントロール等のように、バイナリで提供されているコンポーネントを組み合わせて作成したソフトウェアにおいては、不完全な情報しか与えられず、有効に利用することができない。

【0009】この発明は、上述した事情に鑑みてなされたもので、メモリリークの効果的な検出を可能とすることを目的とする。

【0010】

【課題を解決するための手段】上記目的を達成するため、この発明の第１の観点によるトレースシステムは、アプリケーションプログラムとのインタフェースを取るためのアプリケーションインタフェースを備えるオペレーティングシステムと、前記オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段と、前記アプリケーション手段による前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しを中継するとともに、該呼び出しの情報を取り込むフック手段と、前記フック手段で取り込まれた前記情報を蓄積記録するログ記録手段と、を具備することを特徴とする。

【0011】このシステムによれば、アプリケーションプログラム（ＡＰ）がオペレーティングシステム（ＯＳ）のアプリケーションインタフェース（ＡＰＩ）を呼び出した際に、フック手段がこの呼び出しの情報を取り込み、ログ記録手段がその内容を記録する。従って、例えば、アプリケーションインタフェースの呼び出しが、リソース、例えば、メモリの割り当てや解放に関するものである場合には、割り当て及び解放の履歴がログ記録手段により自動的に生成される。従って、リソースの割り当てと解放の対を判別し、割り当てたが解放されていないものをログから判別することにより、リソースの解放忘れを判別することができる。また、このような構成によれば、アプリケーションプログラムとオペレーティングシステムには、実質的な変更を加えることなく、呼び出し情報の取得が可能となる。

【0012】リソースのリークとしては、例えば、ファイル資源、セマフォ、Mutex等の排他資源、ペン、ブラシ、デバイスコンテキストなどのＧＤＩオブジェクトのリーク等がある。

【００１３】前記フック手段は、例えば、前記アプリケーションインタフェースに対するアプリケーションプログラムの呼び出しを前記アプリケーションインタフェースの代わりに受ける手段と、前記呼び出しの情報を前記ログ記録手段に提供する手段と、前記呼び出しを前記アプリケーションインタフェースに転送する手段とから構成される。

【００１４】前記ログ記録手段で記録された前記呼び出しの情報を出力する手段を配置してもよい。

【００１５】前記フック手段から前記ログ記録手段へ転送された情報の記録のオンとオフを制御するロギング制御手段を配置してもよい。このロギング制御手段は、前記フック手段による前記アプリケーションインタフェースに対する前記呼び出しの転送の停止と実行を、例えば、外部操作に応答して制御する。

【００１６】前記ロギング制御手段は、前記ログ記録手段に記録される前記情報の内容を制御する手段を含んでもよい。

【００１７】前記フック手段は、例えば、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しのうち、前記オペレーティングシステムのリソース割り当てに関するリソース割り当て情報を取り込む。さらに、前記ログ記録手段により、記録されたリソースの割り当てと解放の対を検出し、割り当てと解放の一方のみを検出し、他方を検出できない時に、その旨を報知する対検出手段を配置し、リソースの解放漏れを報知できるようにしてもよい。

【００１８】上記目的を達成するため、この発明の第２の観点によるリソース解放漏れ検出システムは、オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段と、前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムから前記オペレーティングシステムのアプリケーションインタフェースに送られるリソースの割り当てと解放の要求に関するリソース割当情報を取り込むフック手段と、前記フック手段で取り込まれた前記リソース割り当て情報を蓄積記録するログ記録手段と、を具備することを特徴とする。

【００１９】前記ログ記録手段により記録されたリソースの割り当てと解放の対を検出し、割り当てと解放の一方のみを検出し、他方を検出できない時に、その旨を報知する対検出手段を配置しても良い。

【００２０】このようなシステムによれば、オペレーティングシステムとアプリケーションプログラムには、実質的な影響を与えること無く、アプリケーションインタフェースの呼び出しを検出し、この呼び出し情報に基づいてリソースの割り当てと解放の履歴をログに取ることができる。従って、このログをチェックすることにより、リソースの解放忘れを判別することができる。リソースとしては、例えば、ファイル資源、セマフォ、Mutex等の排他資源、ペン、ブラシ、デバイスコンテキストなどのＧＤＩオブジェクト等がある。

【００２１】前記ログ記録手段により記録されたリソースの割り当てと解放の対を検出し、割り当てと解放の一方のみを検出し、他方を検出できない時に、その旨を報知する対検出手段を配置して、リソースの解放漏れを自動的に検出するようにしてもよい。リソースとしてメモリ等が好適である。

【００２２】また、この発明の第３の観点にかかるコンピュータで読み取り可能な記録媒体には、コンピュータを、アプリケーションプログラムとのインタフェースを取るためのアプリケーションインタフェースを備えるオペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段、前記アプリケーション手段による前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムによる前記アプリケーションインタフェースの呼び出しを中継するとともに、該呼び出しの情報を取り込むフック手段と、前記フック手段で取り込まれた前記呼び出しの情報を蓄積記録するログ記録手段、として機能させるためのプログラム（固定データを含む）が記録されている。

【００２３】また、この発明の第４の観点にかかるコンピュータで読み取り可能な記録媒体には、コンピュータを、オペレーティングシステム上でアプリケーションプログラムを実行するアプリケーション手段、前記アプリケーションプログラムの実行に際し、前記アプリケーションプログラムから前記オペレーティングシステムのアプリケーションインタフェースに送られるリソースの割り当てと解放の要求に関するリソース割当情報を取り込むフック手段、前記フック手段で取り込まれた前記リソース割当情報を蓄積記録するログ記録手段、として機能させるためのプログラムが記録されている。

【００２４】

【発明の実施の形態】図１～図６を参照して、この発明の第１の実施の形態に係るトレースシステムについて説明する。

【００２５】この実施の形態は、オペレーティングシステム（ＯＳ）として、ウィンドウズＮＴ及びウィンドウズ９５等を使用し、これらのＯＳにおけるシステムコールがダイナミックリンクライブラリ（Dynamic Link Library：ＤＬＬ～以下、「ＤＬＬ」と称する）によってアプリケーションプログラムとインタフェースしていることに着目し、該ＤＬＬをフック用の特殊なＤＬＬと置き換える。このフック用のＤＬＬによって、全てのアプリケーションインタフェース（Application Program Interface：ＡＰＩ～以下、「ＡＰＩ」と称する）の呼び出しを横取り、すなわちフックして、ＡＰＩの呼び出し元及び時刻等をロギングする。

【００２６】ロギングにより作成されるログファイルを、必要なときに読み込み、これをビューワによってビ

ジュアルに表示する。このようにすることによって、ＯＳのグローバルなトレースを実現し、システムの最適化に必要な、プロセス別及びモジュール別などの情報を得ることができる。さらに、メモリ割り当て及びハンドル割り当て等のＯＳのリソースの割り当てに関する割り当て状況を追跡することによって、メモリリークが発生している箇所を効果的に解析及び表示することが可能となる。このシステムにより、メモリリークの自動的な解析及び表示も実現可能となる。

【００２７】また、メモリに限らず、ファイル資源、セマフォ、Mutex等の排他資源、ペン、ブラシ、デバイスコンテキスト等のＧＤＩオブジェクト等のリークもトレース可能である。

【００２８】図１は、この発明の第１の実施の形態に係るトレースシステムの構成を示している。

【００２９】図１に示すトレースシステムは、アプリケーション部１、第１のフック用ＤＬＬ２、第１の処理用ＤＬＬ３、ログファイル部４、ビューワ５、第２のフック用ＤＬＬ６、第２の処理用ＤＬＬ７、及びコントロールマネージャ８を具備している。アプリケーション部１、第１のフック用ＤＬＬ２、第１の処理用ＤＬＬ３、ビューワ５、第２のフック用ＤＬＬ６、第２の処理用ＤＬＬ７、及びコントロールマネージャ８の各部は、コンポーネントとしてのプログラムモジュールにより構成される。また、ログファイル部４は、記憶装置に記憶されるファイルにより構成される。

【００３０】アプリケーション部１は、ＯＳ（オペレーティングシステム）上で、アプリケーションプログラム（ＡＰ）を実行する。このアプリケーション部１とＯＳとのインタフェースがアプリケーションインターフェース（ＡＰＩ）である。ＯＳのシステムコールは、ＤＬＬによりアプリケーション部１とインタフェースしている。

【００３１】第１のフック用ＤＬＬ２は、アプリケーション部１のＡＰＩ呼び出しをフック（横取り）するためのＤＬＬであり、ＡＰＩ呼び出しに対応する本来のＤＬＬの代わりに挿入される。

【００３２】第１の処理用ＤＬＬ３は、アプリケーション部１のＡＰＩ呼び出しに対応する本来のＤＬＬであり、ＯＳの一部を構成し、該当するＡＰＩ呼び出しに対応する処理を実行する。

【００３３】第１のフック用ＤＬＬ２は、アプリケーション部１の本来のＤＬＬ（第１の処理用ＤＬＬ３）に対する呼び出しをフックし、該呼び出しの情報をログファイル部４に送信するとともに、該呼び出しを第１の処理用ＤＬＬ３に転送する。ログファイル部４は、第１のフック用ＤＬＬ２等でフックされた呼び出し情報をロギング、すなわち記録・蓄積する。

【００３４】ビューワ５は、ログファイル部４のログ内容を読み込み表示する。このビューワ５自体も一種のアプリケーションであるので、ＡＰＩを使用する場合には、第２のフック用ＤＬＬ６を介して、第２の処理用ＤＬＬ７を呼び出す。第２のフック用ＤＬＬ６は、この呼び出しをフックして、その呼び出しの情報をログファイル部４に送る。

【００３５】コントロールマネージャ８は、フック用ＤＬＬ２及び６の動作を制御・管理する。このコントロールマネージャ８を外部操作することにより、呼び出し情報のフックのオン／オフ、フックする呼び出し情報の設定選択、並びに処理用ＤＬＬ３及び７への呼び出しの転送動作のストップ（停止）／ゴー（続行）を制御することができる。

【００３６】図２のフローチャートを参照して、トレースシステムのインストール時の処理の流れを説明する。

【００３７】インストールを開始すると、まず、トレースシステムを適用しようとする処理用ＤＬＬ、例えば処理用ＤＬＬ３及び７を検索する（ステップＳ１）。処理用ＤＬＬ３及び７が検索されると、検索された処理用ＤＬＬ３及び７の原ファイル名が他のダミーファイル名にリネームされる（ステップＳ２）。

【００３８】次に、フック用ＤＬＬ２及び６が先にリネームした処理用ＤＬＬの原ファイル名でシステムに導入される（ステップＳ３）。そして、ログファイルを参照するためのログファイルビューワがシステムに組み込まれ（ステップＳ４）、さらにフック用ＤＬＬ２及び６を制御するためのコントロールマネージャ８がシステムに組み込まれる（ステップＳ５）。

【００３９】上述の処理について、ここでは、ＯＳとして、マイクロソフト社のウィンドウズＮＴシステムを使用するものとし、該トレースシステムのＤＬＬのフックシステムをバイルスシステム（ＶＩＲＵＳシステム）と称することにする。

【００４０】この場合、バイルスシステムがフックするＤＬＬは、３２ビットグラフィックデバイスインタフェース（Graphic Device Interface：ＧＤＩ～以下、「ＧＤＩ」と称する）用ＤＬＬであるＧＤＩ３２．ＤＬＬ、３２ビットユーザ用ＤＬＬであるＵＳＥＲ３２．ＤＬＬ、ウィンドウズＮＴ用ＤＬＬ処理のためのＤＬＬであるＮＴＤＬＬ．ＤＬＬ及び３２ビットＯＬＥ用のＤＬＬであるＯＬＥ３２．ＤＬＬの４種類からなるデフォルトの３２ビットウィンドウズ用ＤＬＬ（以下、「ＷＩＮ３２ＤＬＬ」と称する）のうちのＧＤＩ３２．ＤＬＬ、ＵＳＥＲ３２．ＤＬＬ及びＮＴＤＬＬ．ＤＬＬである。

【００４１】これらＧＤＩ３２．ＤＬＬ、ＵＳＥＲ３２．ＤＬＬ及びＮＴＤＬＬ．ＤＬＬは、メモリリークの検出に使用することができる。ちなみに、これらのＧＤＩ３２．ＤＬＬ、ＵＳＥＲ３２．ＤＬＬ及びＮＴＤＬＬ．ＤＬＬは、基本的な３２ビットウィンドウズシステムにおけるグローバルトレースにも使用することができ、ＯＬＥ３２．ＤＬＬは、グローバルトレース用にの

み使用することができる。

【0042】図3は、バイルスシステムを導入する前のGDI32．DLLに関するモジュール構成を示し、図4は、バイルスシステム導入後のGDI32．DLLに関するモジュール構成を示している。

【0043】導入に際して、バイルスシステムは、デフォルトのWIN32．DLLであるGDI32．DLL等の上述した3つのDLLファイルを、まず、別のダミーのファイル名、例えばGDI32．DLLをVIRUSGDI．DLLにリネームする。

【0044】次に、フックによるロギング機構を持つバイルスシステム用のDLL、すなわちフック用DLLを、GDI32．DLLとしてシステムに導入し、先にリネームしたもとのGDI32．DLLと置換する。このフック用DLLは、GDI32．DLLと同じ外部インタフェース及びパラメータ構成を有しているが、予め設定した特定のAPIに対しては、API呼び出しに関する情報をログファイル部4にロギングするように作成されている。つまり、アプリケーションプログラムは、もとの処理用DLLであるGDI32．DLLとリンクしているものとして動作するが、実際には、バイルスシステムのフック用DLLとリンクしていることになる。バイルスシステムのフック用DLLは、ロギング後、VIRUSGDI．DLLとして、本来のGDI32．DLLを呼び出し、必要な情報を提供する。このようにして、GDI呼び出し情報をフックする。

【0045】上述したGDI32．DLL、USER32．DLL及びNTDLL．DLLのような通常のDLLは、処理実体である実体オブジェクトがDLLそのものに存在する。従って、これらのDLLは、アプリケーションプログラムから呼び出されると、エントリポイントだけをフックすれば、容易にロギングを行うログ機構を構築することができる。

【0046】次に、図5に示すフローチャートを参照して、アプリケーションプログラムの実行時における動作を説明する。アプリケーション部1でアプリケーションが実行されると、アプリケーションは必要に応じて、API呼び出しを行う（ステップS11）。このAPI呼び出しは、第1のフック用DLL2でフックされる（ステップS12）。第1のフック用DLL2は、フックした呼び出しの情報をログファイルにロギングする（ステップS13）。第1のフック用DLL2は、アプリケーション部1からの呼び出しに応じて、本来の第1の処理用DLL3を呼び出す（ステップS14）。第1の処理用DLL3は、該当するAPI処理を行う（ステップS15）。

【0047】アプリケーションプログラムの実行が終了したか否かが判定され（ステップS16）、アプリケーションプログラムの実行中は、API呼び出し毎に、上述したステップS11～S15の処理が繰り返される。

このようにして、API呼び出し情報がロギングされ、ログファイル部4にAPI呼び出し情報の履歴すなわちログが記録される。

【0048】すなわち、バイルスシステムでは、APIの呼び出し時及び処理後のリターン時に、ロギング処理を行い、呼び出したモジュール名、アドレス、時刻、及びパフォーマンスカウント値等をログファイル部4に記録する。このようにすることにより、あるポイントから、あるポイントまでの処理時間を計測することができる。この場合、通常の局所的な計測とは異なり、OSのライブラリを置き換えているため、測定対象となるモジュールには、一切、変更を加える必要がない。このようにすることによって、処理時間のボトルネックを容易に調べることができる。

【0049】さらに、メモリリーク等の、リソースの解放漏れの検出は、このようなトレースシステムと、図6に示すように、解析部9とを組み合わせて実現する。解析部9は、例えば、在来のフューザに機能追加することによりソフトウエアで実現することができ、ログファイル部4に記録されたログの内容をトレースし、メモリの割り当てと解放を行ったモジュール名の対を検出し、割り当てと解放が対になっていないモジュールをリストアップし、表示すると共に警告を発する。

【0050】即ち、メモリリークは、メモリ割り当ての呼び出しがあるにもかかわらず、メモリが解放されない場合に発生する。従って、モジュール名に着目してログをトレースし、割り当てと解放が対になっていないモジュールをリストアップすることにより判別することができる。まれに、メモリを割り当てるモジュールと解放を行うモジュールが別々の場合もあるが、解放アドレス及びハンドル値もトレースし、これらも表示することにより、正しいリーク情報を表示することができる。解放と割り当てが別々の場合は、まれにプログラムのバグであることも考えられるため、警告情報として出力する。

【0051】このようにして、ウィンドウズNT及びウィンドウズ95等のOSのAPIの呼び出しをフック用DLL2及び6等によってフックして、APIの呼び出し元及び時刻等をロギングすることができる。ログファイル部4をビューワによってビジュアルに表示することによって、OSのグローバルなトレースを実現して、システムの最適化に必要な、プロセス別及びモジュール別などの情報を得ることができる。これらの情報に基づいて、システムのパフォーマンスチューニングを実現することができる。また、メモリ割り当て及びハンドル割り当て等のOSのリソースの割り当て状況を追跡することによって、メモリリークが発生している箇所を効果的に解析及び表示することが可能となる。このシステムを用いることにより、メモリリークの自動的な解析及び表示を実現することもできる。

【0052】図7は、この発明の第2の実施の形態に係

るトレースシステムの構成を示している。図7は、ＯＬＥインタフェースのように、処理用ＤＬＬ自体に処理用のオブジェクトが埋め込まれておらず、別の実体モジュールにリンクしていて、該実体モジュールを呼び出して処理を行うものを含んでいる場合のトレースシステムを示している。

【００５３】図7のトレースシステムは、アプリケーション部１１、第1のフック用ＤＬＬ１２、第1の処理用ＤＬＬ１３、実体モジュール１４、ログファイル部４、ビューワ５、第2のフック用ＤＬＬ６、第2の処理用ＤＬＬ７、及びコントロールマネージャ８を具備している。

【００５４】アプリケーション部１１、第1のフック用ＤＬＬ１２及び第1の処理用ＤＬＬ１３は、図1のアプリケーション部１、第1のフック用ＤＬＬ２及び第1の処理用ＤＬＬ３とほぼ同様であるが、第1の処理用ＤＬＬ１３が実体モジュール１４を呼び出して処理を実行させる。すなわち、第1の処理用ＤＬＬ１３は、その一部又は全部のインタフェースに関して、第1の処理用ＤＬＬ１３自体で処理を実行せず、実体モジュール１４に処理を実行させる。

【００５５】第1のフック用ＤＬＬ１２が挿入されない場合、第1の処理用ＤＬＬ１３が、最初の呼び出し時に、アプリケーション部１１に実体モジュール１４のエントリポイントを返し、以後は、アプリケーション部１１から、実体モジュール１４が直接呼び出される。そこで、第1のフック用ＤＬＬ１２は、実体モジュール１４のエントリポイントに代えてダミーのエントリポイントとして、第1のフック用ＤＬＬ１２のエントリポイントを返す。このようにして、当該インタフェースを、アプリケーション部１１が呼び出すたびに、第1のフック用ＤＬＬ１２を通って実体モジュール１４が呼び出される。したがって、実体モジュール１４の呼び出しの直前にロギングすることが可能になる。

【００５６】ログファイル部４、ビューワ５、第2のフック用ＤＬＬ６、第2の処理用ＤＬＬ７、及びコントロールマネージャ８については、図1の場合と同一である。アプリケーション部１１、第1のフック用ＤＬＬ１２、第1の処理用ＤＬＬ１３、実体モジュール１４、ビューワ５、第2のフック用ＤＬＬ６、第2の処理用ＤＬＬ７、及びコントロールマネージャ８の各部は、コンポーネントとしてのプログラムモジュールにより構成される。また、ログファイル部４は、記憶装置に記憶されるファイルにより構成される。

【００５７】第1の処理用ＤＬＬ１３が処理を実体モジュール１４に依存せずに、それ自体で処理を実行する場合の、第1のフック用ＤＬＬ１２及び第1の処理用ＤＬＬ１３の機能及び動作は、図1における第1のフック用ＤＬＬ２及び第1の処理用ＤＬＬ３と全く同様となる。

【００５８】次に、図7のトレースシステムのアプリケーションの実行動作について具体的に説明する。図7のトレースシステムのインストールは、図1のシステムと同様に図2に示した流れに従って行われる。

【００５９】前述のように、例えば、ＯＬＥのモジュールであるＯＬＥ２３．ＤＬＬの場合には、通常のＤＬＬと比して若干特殊な動作をする。すなわち、ＯＬＥはＤＬＬで実現されているものの、通常のＤＬＬとは、インタフェースが異なっている。ＯＬＥ３２．ＤＬＬは、アプリケーションプログラムから呼び出されると、処理実体である実体モジュール１４を呼び出して、実体モジュール１４により処理を実行させる。したがって、ロギングのためのログ機構の構築に工夫が必要である。

【００６０】先に述べたように、通常のＤＬＬは、アプリケーション部１１で実行されるアプリケーションプログラムから呼び出されると、処理実体がＤＬＬそのものに埋め込まれて存在し、単に、エントリポイントだけをフックするだけで容易にロギングすることができる。

【００６１】ところが、ＯＬＥのインタフェースは、最初の呼び出しは、ＯＬＥのＤＬＬを通るが、それが実体モジュールのポインタを返すため、次回以降の呼び出しは、アプリケーションプログラムからＤＬＬを経由せず実体モジュールが呼び出される。

【００６２】そこで、ＯＬＥのインタフェースでは、単にＯＬＥのＤＬＬをフックしただけでは不充分であり、第1のフック用ＤＬＬ１２において、ポインタとして、ダミーのインタフェース、つまり、バイルスシステムのＤＬＬである第1のフック用ＤＬＬ１２のエントリポイントを返す。そして、第1のフック用ＤＬＬ１２が実体モジュール１４を呼び出すようにすることによって、実体モジュール１４を呼び出す直前にロギング等を行うことが可能となる。

【００６３】次に、図8に示すフローチャートを参照して、アプリケーションの実行時における動作を説明する。アプリケーション部１１でアプリケーションが実行されると、ＡＰＩ呼び出しが行われる（ステップＳ２１）。ＡＰＩ呼び出しは、第1のフック用ＤＬＬ１２でフックされ（ステップＳ２２）、フックされた呼び出しの情報をログファイル部４にロギングする（ステップＳ２３）。第1のフック用ＤＬＬ１２は、アプリケーション部１１からの呼び出しに応じて第1の処理用ＤＬＬ１３を呼び出し（ステップＳ２４）、呼び出しがＯＬＥインタフェースか否かが判定される（ステップＳ２５）。ＯＬＥでなければ、第1の処理用ＤＬＬ１３において、該当するＡＰＩ処理を行う（ステップＳ２６）。

【００６４】その後、アプリケーション部１１におけるアプリケーションプログラムの実行が終了したか否かが判定される（ステップＳ２７）。ステップＳ２５で、ＯＬＥインタフェースであると判定されると、第1のフック用ＤＬＬ１２でエントリポイントが変換されて（ステップＳ２８）、実体モジュール１４が呼び出される。実

体モジュール14で該当するAPI処理が行われ（ステップS29）、ステップS27に移行する。なお、一旦、ステップS28でエントリ変換が行われた後は、ステップS23のロギング後、第1のフック用DLL12から、第1の処理用DLL13をその都度呼び出さずに、実体モジュール14を直接呼び出すようにしてもよい。

【0065】ステップS27の判定に応じ、アプリケーションプログラムの実行中は、API呼び出し毎に、上述したステップS21～S26又はS21～S25→S28→S29の処理が繰り返される。このようにして、API呼び出し情報がロギングされ、ログファイル部4にAPI呼び出し情報の履歴すなわちログが記録される。

【0066】なお、図8では、ステップS25において、OLEか否かの判別を行っているが、個々のDLLを置き換える際に、ダミーのDLL（OSに付属しているものではなく、ロギング取得用に作成したもの）のロジックの中にOLEを埋め込んでおき、OLEか否かの判断を行わないようにしてもよい。

【0067】上述したOLEのモジュールの場合について、さらに詳細に説明する。前述のように、OLEはDLLで実現されているものの、通常のDLLとは、インタフェースが少し異る。図9に示すように、通常のDLLは、アプリケーションプログラムAPから呼び出されると処理実体がDLLそのものに存在し、単に、エントリポイントだけをフックするようにすれば、ログ機構などを挿入することが容易である。

【0068】ところが、図9のように、OLEのインタフェースは、最初の呼び出しは、OLEのDLLを通るが、それが実体モジュールのエントリのポインタを返すため、次回以降の呼び出しは、アプリケーションプログラムAPからDLLを経由せずに実体モジュールが呼び出される。従って、OLEのインタフェースでは、フック用DLLで、単にOLEのDLLをフックしただけでは不充分であり、図10に示すように、エントリのポインタとして、ダミーのインタフェース、つまり、バイルスシステムのDLL（フック用DLL）のエントリポイントを返すようにする。該ポインタによって実体モジュールを呼び出すようにすることで、実体モジュールを呼び出す直前にロギング等を行うことが可能となる。

【0069】即ち、OLEインタフェース（OLEのインタフェース）は、C++のインプリメントと同様に、CoCreateInstanceにて、ポインタを取得後、このポインタを元にしてそれぞれのインタフェースを呼び出す。バイルスシステムでは、最初のCoCreateInstanceは、通常のDLL同様、DLLインタフェースのフックにより、本来のOLE32．DLLを呼び出すが、このとき返すオブジェクトポインタを、OLE32．DLLが返すポインタでなく、自分自身のダミーのポインタに返す。すると、OLEを呼び出したアプリケーションは、ダミーのポインタを本来のポインタとして取り扱うため、次にオブジェクトを呼び出す時は、このダミーのポインタを元にオブジェクトを呼び出す。このポインタは、バイルスシステムが払い出したポインタであり、アプリケーションがオブジェクトを呼び出す度に、バイルスシステムでフックでき、1通りのロギング動作を行ってから、本来のOLE32．DLLが返したポインタを呼び出す。これらの点が通常のDLLと動作が異なる点である。

【0070】既に述べたように、バイルスシステムは、APIの呼び出し時及び処理終了後のリターン時に、ロギング処理を行い、呼び出したモジュール名、アドレス、時刻及びパフォーマンスカウント値等をログファイルに記録する。このようにして、あるポイントから、あるポイントまでの処理時間を計測することができる。通常の局所的な計測とは異なり、OSのライブラリを置き換えているため、測定対象となるモジュールは一切変更する必要がない。また、メモリリークの検出には、図6に示したように、この仕組みと、解析部9とを組み合わせて使用する。

【0071】図1及び図7に示したトレースシステムを、さらに具体的に説明する。このシステムは、主として図11に示すような、コンポーネントモジュールで構成される。

【0072】図11は、GAP．EXE、GAPUI．DLL、GAPUI．OCX、FSENGINE．DLL、VRSCTRL．EXE、NOSIDEIF．DLL、VIRUSTUB．DLL及びVIRUSLOG．DLLの各モジュールを示す。

【0073】GAP．EXEは、ビューワの実行モジュールであり、ビューワ5に相当する。GAPUI．DLLは、GAP．EXE特有のインタフェース用DLL、GAPUI．OCXは、GAP．EXE用のOCX（OLEコントロール）21である。

【0074】VRSCTRL．EXEは、コントロールマネージャの実行モジュールであり、コントロールマネージャ8に相当する。VIRUSTUB．DLL及びVIRUSLOG．DLLは、バイルスシステムのAPI呼び出し情報をフックするためのDLLであり、これらはフック用DLL2、6又は12に相当する。GAPUI．DLL、FSENGINE．DLL及びNOSIDEIF．DLLは、各種処理を実行するためのDLLであり、処理用DLL3、7又は13に相当する。

【0075】図11に示した、コンポーネントは、主として、次の4つのサービス提供する。

(1) カレント情報取得サービス

現在のGDIオブジェクトの割り当ての監視情報、並びにOLE、GDI等のAPI呼び出しの発行状況の監視情報を、ロギングバッファ及びログファイル部4の少なくともいずれかから取得し、ビューワ5であるGAP．

EXEにより表示する。GDIオブジェクトの割り当ての監視情報は、リソースの割り当てと開放、及びオーナの表示情報を含む。

【0076】(2) ロギングサービス

GDI及びOLE用のDLLの一部のインターフェースについて、VIRUSTUB.DLLによるパッチを当てる。具体的には、上述したように、インストール時に本来の処理用DLLをリネームして、代わりに、処理用のDLLのファイル名としてフック用のダミーのDLLを挿入する。

【0077】このようにすることにより、必要なインタフェースが呼び出されるたびに、VIRUSのDLLを経由することになり、呼び出し、オブジェクトの割り当て、及び開放状況がすべて把握できる。この機能では、DLL及び実行プログラム（以下、「EXE」と称する）毎にロギング機構をオン／オフ設定することができ、ロギングレベルも例えば0＝OFF、1＝呼び出しのみ、及び2＝呼び出しとパラメータ等の3段階に設定することができる。ロギングにより、ログファイル等に取得された情報は、フューザで表示させることができる。

【0078】(3) ダイナミックエグゼキューションサービス

このサービスは、ロギングサービスを利用して実現する。アプリケーションプログラムがOLE及びGDIを呼び出す時に、コントロールマネージャ8であるVRSCTRL、EXEからの指示で、ストップ／ゴーを指示することができる。この機能は、デバッガを経由することなしに、呼び出しパラメータを確認したり、OLEの内部情報、及びGDIのリソース割り当て状況等を表示させるときに使用する。

【0079】(4) デバッギングライブラリサービス

プログラム言語、例えばビジュアルBASIC（Visual Basic：VB）等の開発者用には、デバッグサポートライブラリが提供される。

【0080】基本的な動作は、次のようになる。ウィンドウズNT及びウィンドウズ95等において、標準でサポートされるAPI用のDLLを、このバイルスシステムのVIRUSTUB.DLLで置換する。このDLLでは、全ての標準インタフェースのエントリポイントだけが定義されている。一方、既存のDLLは、リネームする。

【0081】アプリケーションからのAPI呼び出しは、既存のライブラリではなく、置換されたバイルスシステムのDLLを経由するようになる。一方、リネームされた既存のDLLには、本来のAPIの機能が含まれているため、ロギングした後は、そのライブラリを呼び出すようにする。本来の処理後、再びバイルスシステムのDLLに処理が戻ってくるため、その間の時間をロギングすれば、処理時間などを把握することができる。

【0082】また、リソースの割り当ては、確保と開放の呼び出しが存在するため、それらが対になっていないときは、リークが発生したと判断して、ビューワで警告を与えるようにする。リソースは、モジュール名、ハンドル値及びアドレスなどによって特定することができる。

【0083】このようにして、ウィンドウズNT及びウィンドウズ95等のOSにおけるAPI呼び出しをフック用DLL12及び6等によってフックして、APIの呼び出し元及び時刻等をロギングすることができる。ログファイル部4をビューワ5によってビジュアルに表示することによって、OSのグローバルなトレースを実現して、システムの最適化に必要な、プロセス別及びモジュール別などの情報を得ることができる。しかも、リコンパイルや再リンク等の操作も不要である。

【0084】上述のトレースシステムは、専用のシステムによらず、通常のコンピュータシステムを用いて実現可能である。例えば、コンピュータに上述の動作を実行するためのプログラムを記録した媒体（フロッピーディスク、CD-ROM等）から該プログラムをインストールすることにより、上述の処理を実行するシステムを構成することができる。

【0085】また、コンピュータにプログラムを供給するための媒体は、通信媒体（通信回線、通信ネットワーク、通信システムのように、一時的に流動的にプログラムを保持する媒体）でも良い。例えば、通信ネットワークの掲示板（BBS）に該プログラムを掲示し、これをネットワークを介して配信してもよい。そして、このプログラムを起動し、OSの制御下で、他のアプリケーションプログラムと同様に実行することにより、上述の処理を実行することができる。

【0086】なお、OSが上述のシステムの機能の一部を担当する場合、或いは、OSとアプリケーションプログラムが共同して上述の機能の一部又は全部を達成する場合には、記録媒体にはOS以外のプログラム部分又はデータ部分を格納しておけばよい。

【0087】

【発明の効果】以上説明したように、この発明のトレースシステム及びメモリリーク検出システムでは、アプリケーションプログラムのインタフェース呼び出しのたびに、その呼び出しに係る呼び出し情報やリソース割り当て情報をフックしてログ記録するので、オペレーティングシステムのグローバルなトレース及びメモリリーク等のリソースの解放漏れの効果的な検出を行うことができる。すなわち、この発明では、リコンパイルが不要で、バイナリで提供されたコンポーネントを組み合わせて作成されたシステムにおいても、OSのグローバルなトレース及びそれによるメモリリークの効果的な検出を可能とし、メモリリークの検出及び解決を始めとするシステムのパフォーマンスチューニングに効果的に利用し得る

トレースシステム及びメモリリーク検出システムを提供することができる。

【図面の簡単な説明】

【図1】この発明の第1の実施の形態に係るトレースシステムの構成を模式的に示すブロック図である。

【図2】図1のトレースシステムにおけるインストール動作を説明するためのフローチャートである。

【図3】図1のトレースシステムのＤＬＬの動作原理を説明するための模式図である。

【図4】図1のトレースシステムのフックの原理を説明するための模式図である。

【図5】図1のトレースシステムにおけるアプリケーションプログラムの実行動作を説明するためのフローチャートである。

【図6】メモリリーク等のリソースの解放漏れを検出するシステムの構成を示すブロック図である。

【図7】この発明の第2の実施の形態に係るトレースシステムの構成を模式的に示すブロック図である。

【図8】図7のトレースシステムにおけるアプリケーションプログラムの実行動作を説明するためのフローチャートである。

【図9】図7のトレースシステムの通常のＤＬＬと、ＯＬＥのＤＬＬの動作原理を説明するための模式図である。

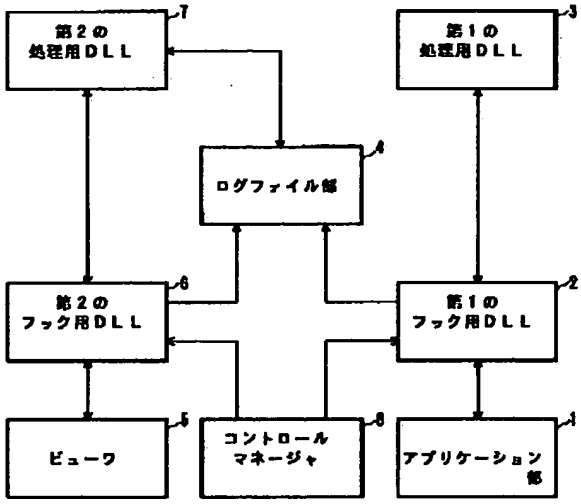
【図10】図7のトレースシステムのＯＬＥのＤＬＬのフックの原理を説明するための模式図である。

【図11】図1及び図7のトレースシステムにおける具体的なモジュール構成の一例を示す模式図である。

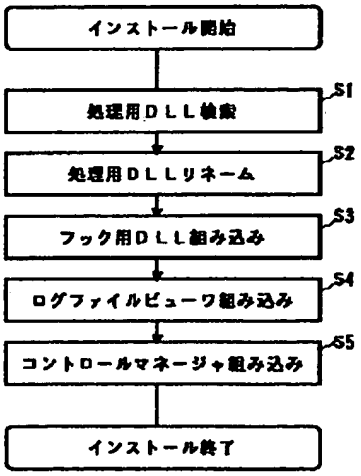
【符号の説明】

1，11	アプリケーション部
2，6，12	フック用ダイナミックリンクライブラリ（ＤＬＬ）
3，7，13	処理用ダイナミックリンクライブラリ（ＤＬＬ）
4	ログファイル部
5	ビューワ
8	コントロールマネージャ
14	実体モジュール

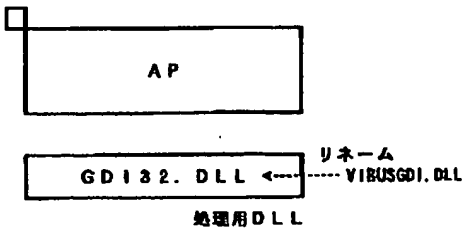
【図1】



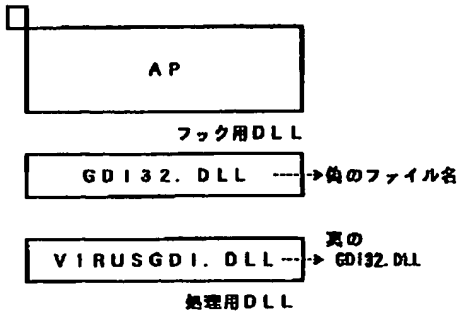
【図2】



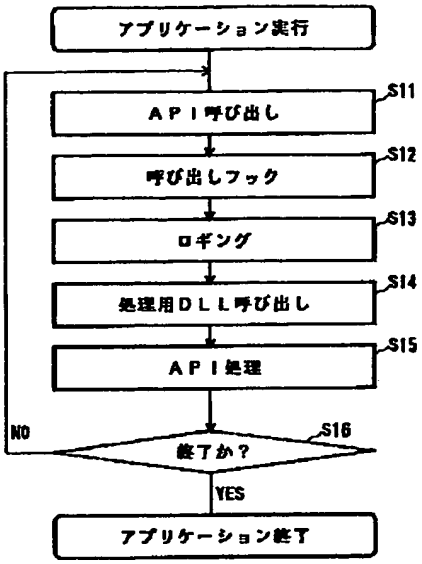
【図3】



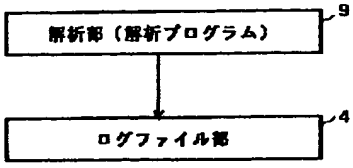
【図4】



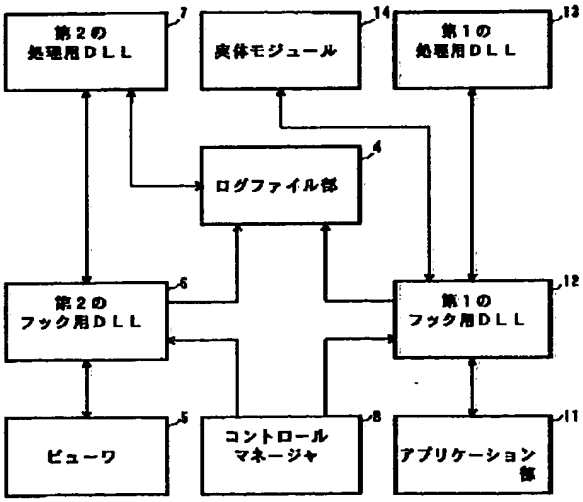
【図5】


アプリケーション実行
S11
ＡＰＩ呼び出し
S12
呼び出しフック
S13
ロギング
S14
処理用ＤＬＬ呼び出し
S15
ＡＰＩ処理
S16
終了か？
NO
YES
アプリケーション終了

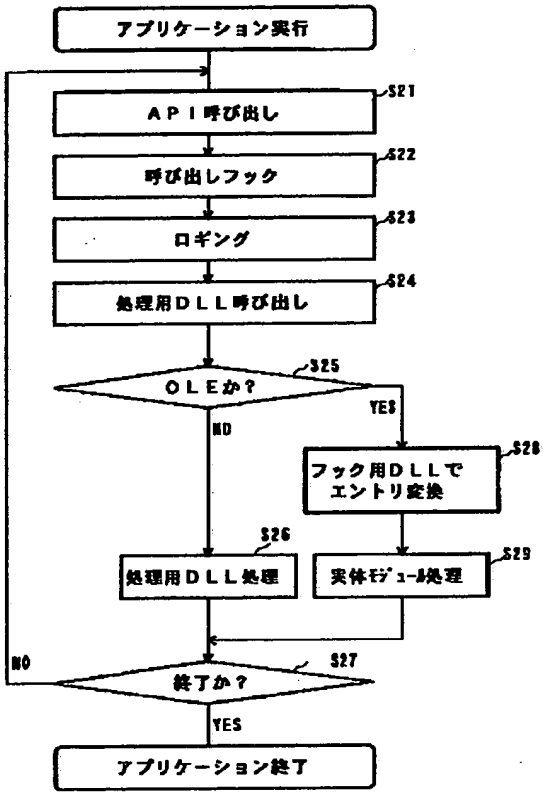
【図6】


解析部（解析プログラム）
9
ログファイル部
4

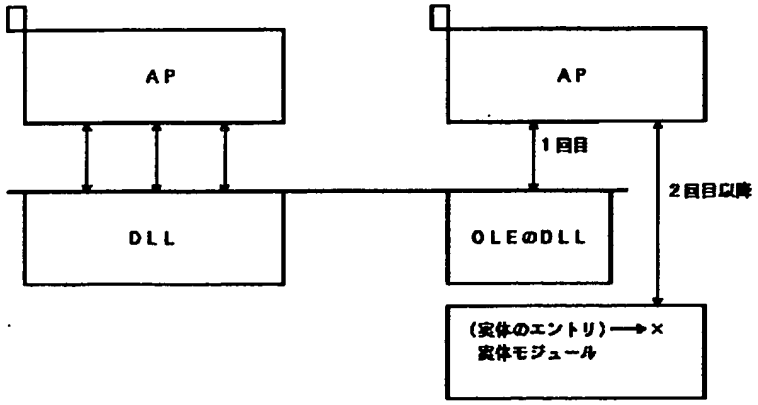
【図7】


第２の
処理用ＤＬＬ
7
実体モジュール
14
第１の
処理用ＤＬＬ
13
ログファイル部
4
第２の
フック用ＤＬＬ
6
第１の
フック用ＤＬＬ
12
ビューワ
5
コントロール
マネージャ
8
アプリケーション
部
11

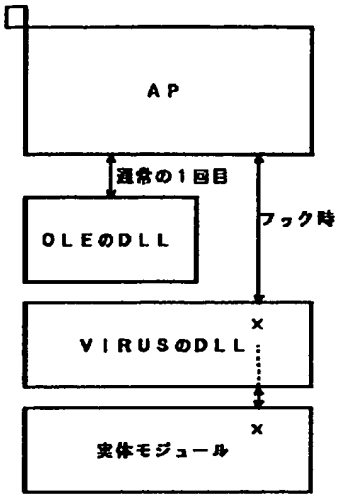
【図8】


アプリケーション実行
S21
ＡＰＩ呼び出し
S22
呼び出しフック
S23
ロギング
S24
処理用ＤＬＬ呼び出し
S25
ＯＬＥか？
YES
NO
S28
フック用ＤＬＬで
エントリ変換
S26
処理用ＤＬＬ処理
S29
実体ﾓｼﾞｭｰﾙ処理
S27
終了か？
NO
YES
アプリケーション終了

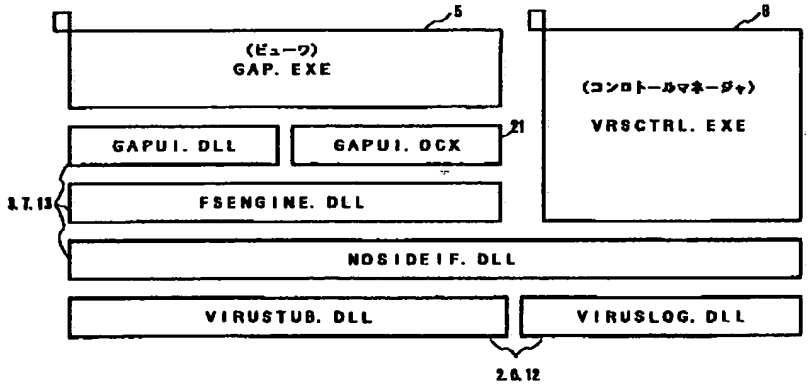
【図9】


AP
AP
1回目
2回目以降
DLL
OLEのDLL
(実体のエントリ)→×
実体モジュール

【図10】


AP
通常の1回目
フック時
OLEのDLL
VIRUSのDLL
×
実体モジュール
×

【図11】


5
(ビューワ)
GAP. EXE
8
(コントロールマネージャ)
VRSCTRL. EXE
GAPUI. DLL
GAPUI. OCX
21
3.7.13
FSENGINE. DLL
NDSIDEIF. DLL
VIRUSTUB. DLL
VIRUSLOG. DLL
2.6.12